AUSG

네? 송금시스템을 람다에 올린다고요?

8기 이현제

AUSG

안녕하세요!

- AUSG 8기
- 터키어와 소프트웨어를 전공했어요
- 온라인 주문서 플랫폼 스타트업 백엔드
- AWS도 관리, 6개월차 응애

AUSG

안녕하세요!

- AUSG 8기
- 터키어와 소프트웨어를 전공했어요
- 온라인 주문서 플랫폼 스타트업 백엔드
- AWS도 관리, 6개월차 응애
- 서버리스를 적극 사용 중

AUSG

AUSG

일회용 입금자명이 발급돼요

스마트 무통장에서는 [수령자명+숫자] 조합으로 일회용 입금자명이 발급돼요!

'입금자명'을 '일회용입금자명'으로 변경해서 입금해주셔야 정상적으로 입금 확인 및 주문이 완료됩니다.
송금하시는 은행 앱에서 '받는 분 통장에 표시' 부분에 일회용 입금자명을 입력해 주세요.

AUSG

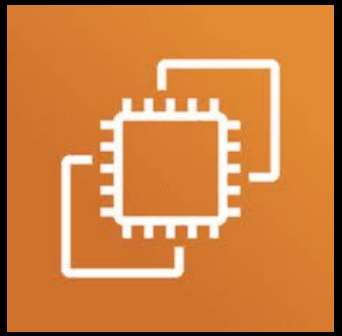
요구사항

- 5분마다 전체 로직이 도는 반 리얼타임 서비스
- 다운 타임이 없어야 한다 -> 절대 죽어서는 안된다!
- 비용이 적었으면 좋겠다 -> 운영비용 / 서버비용
- 다른 AWS 서비스와 통합이 잘 됐으면 좋겠다

AUSG

고민..

ECS



EC2

Lambda

AUSG

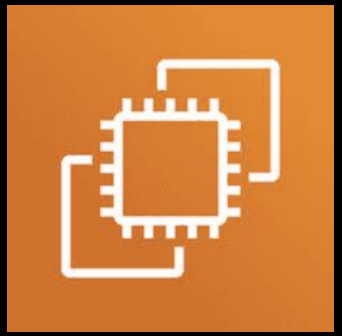
고민..

- 5분마다 전체 로직이 도는 반 리얼타임 서비스
- 다운 타임이 없어야 한다 -> 절대 죽어서는 안된다!
- **비용이 적었으면 좋겠다 -> 운영비용 / 서버비용**
- 다른 AWS 서비스와 통합이 잘 됐으면 좋겠다

AUSG

고민..

ECS

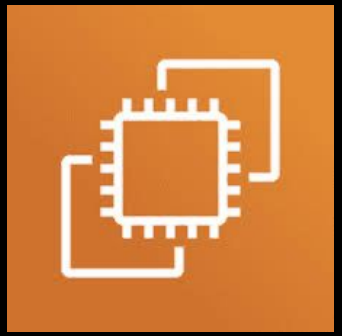


EC2

Lambda

결정!

ECS



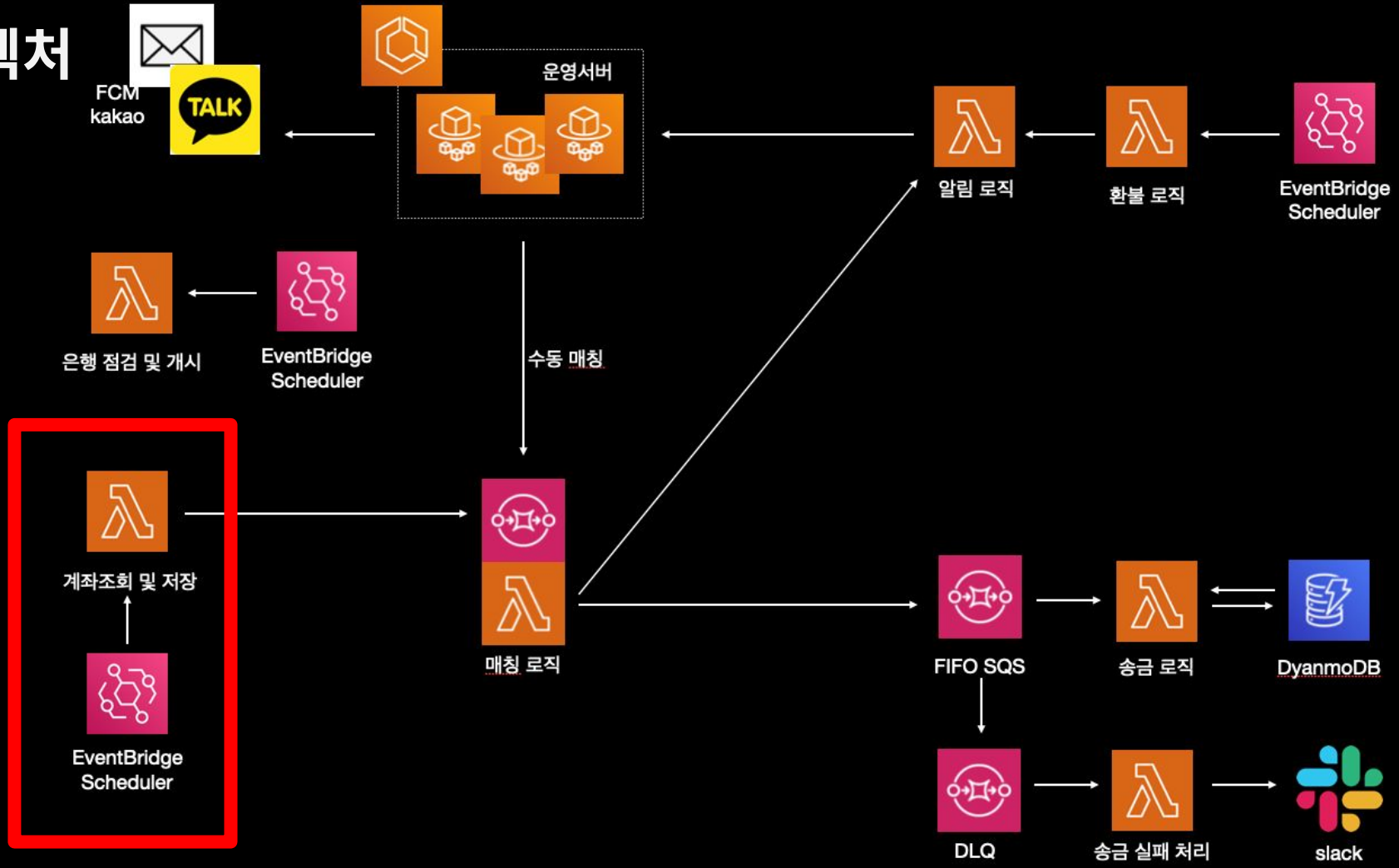
EC2

Lambda

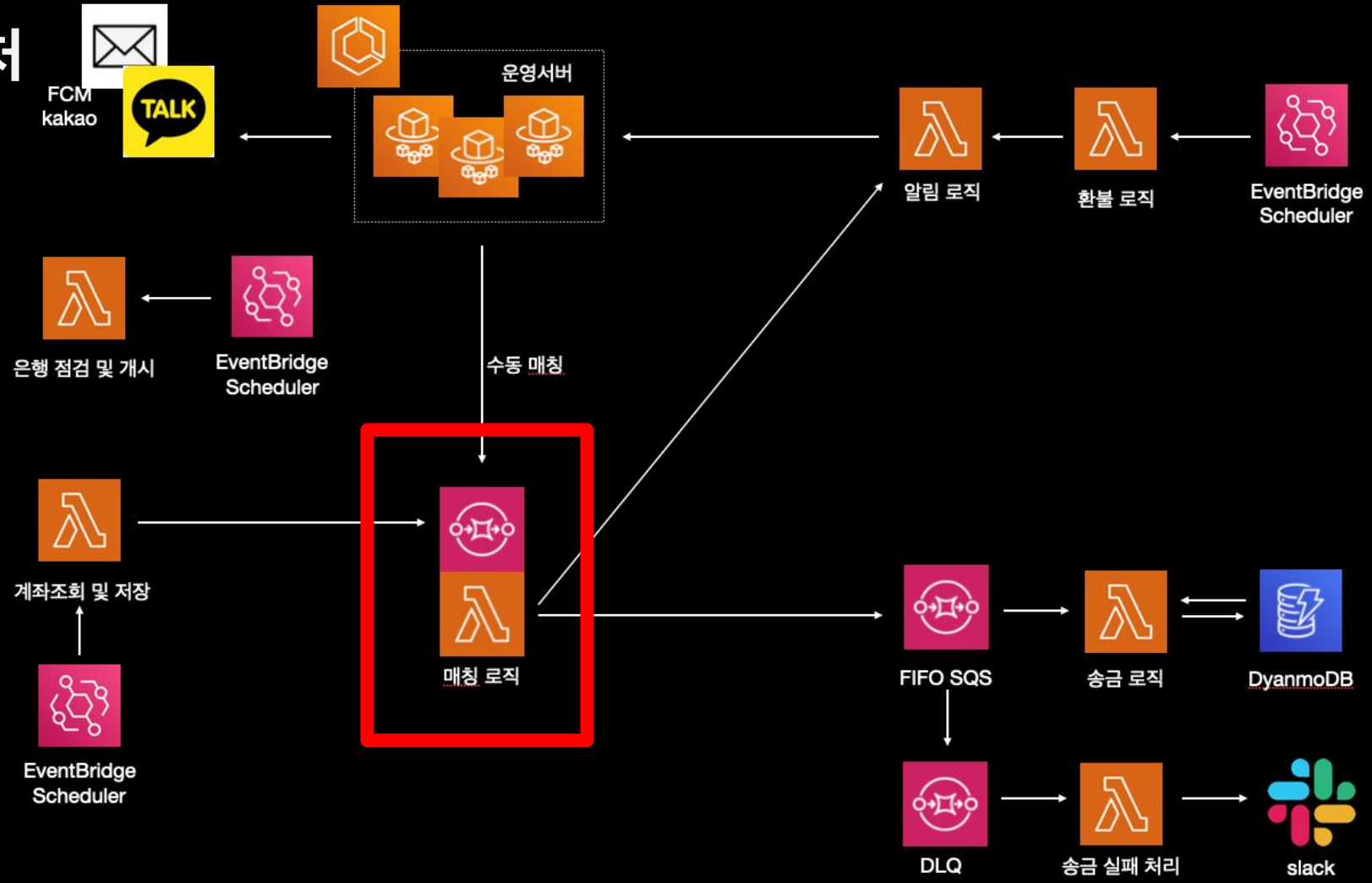
죽을 일이 없다 -> 운영비용 적다

1백만건 당 0.20 USD 🤑👍

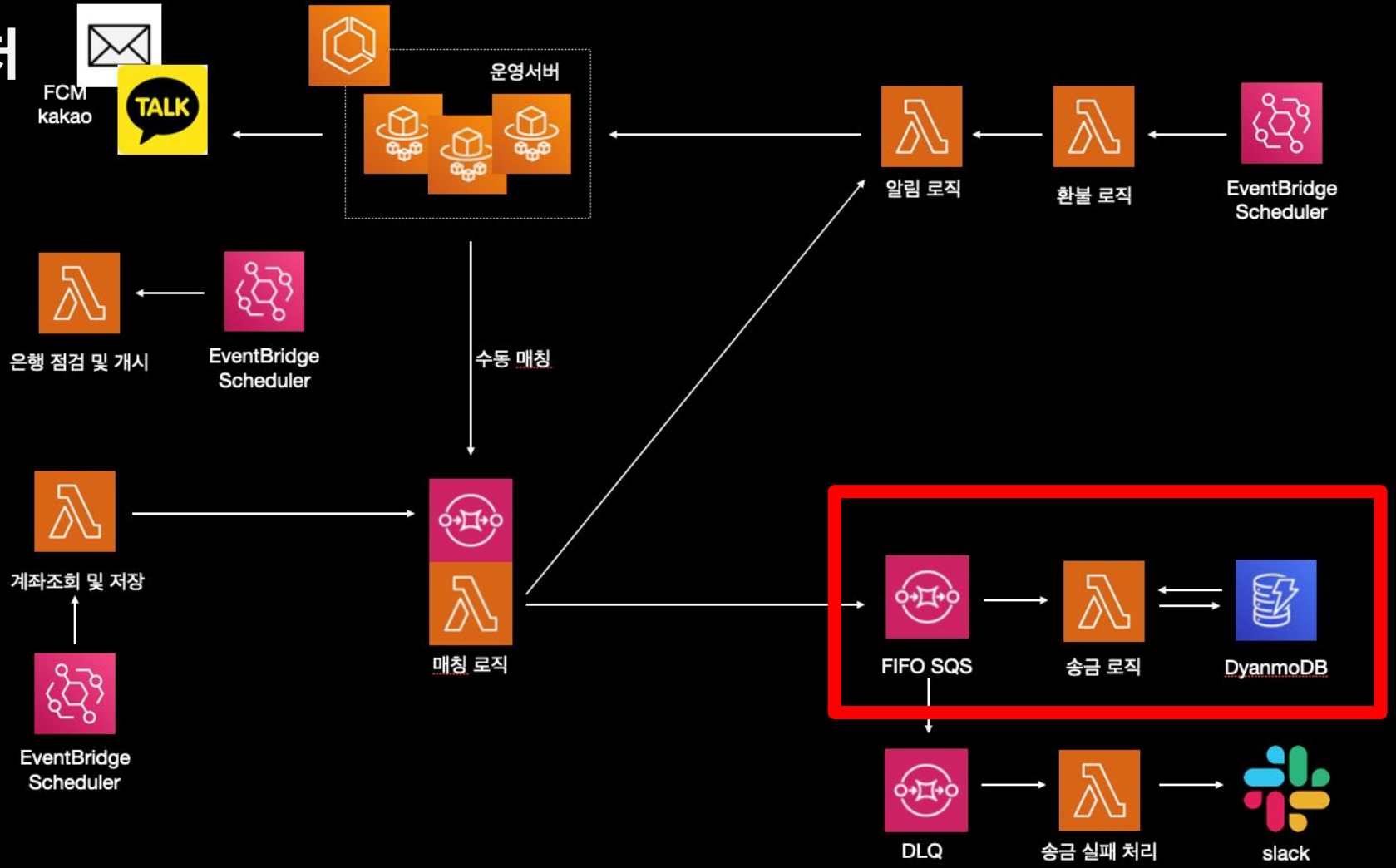
아키텍처


FCM
kakao
TALK
운영서버
알림 로직
환불 로직
EventBridge
Scheduler
은행 점검 및 개시
EventBridge
Scheduler
수동 매칭
계좌조회 및 저장
EventBridge
Scheduler
매칭 로직
FIFO SQS
송금 로직
DyanmoDB
DLQ
송금 실패 처리
slack

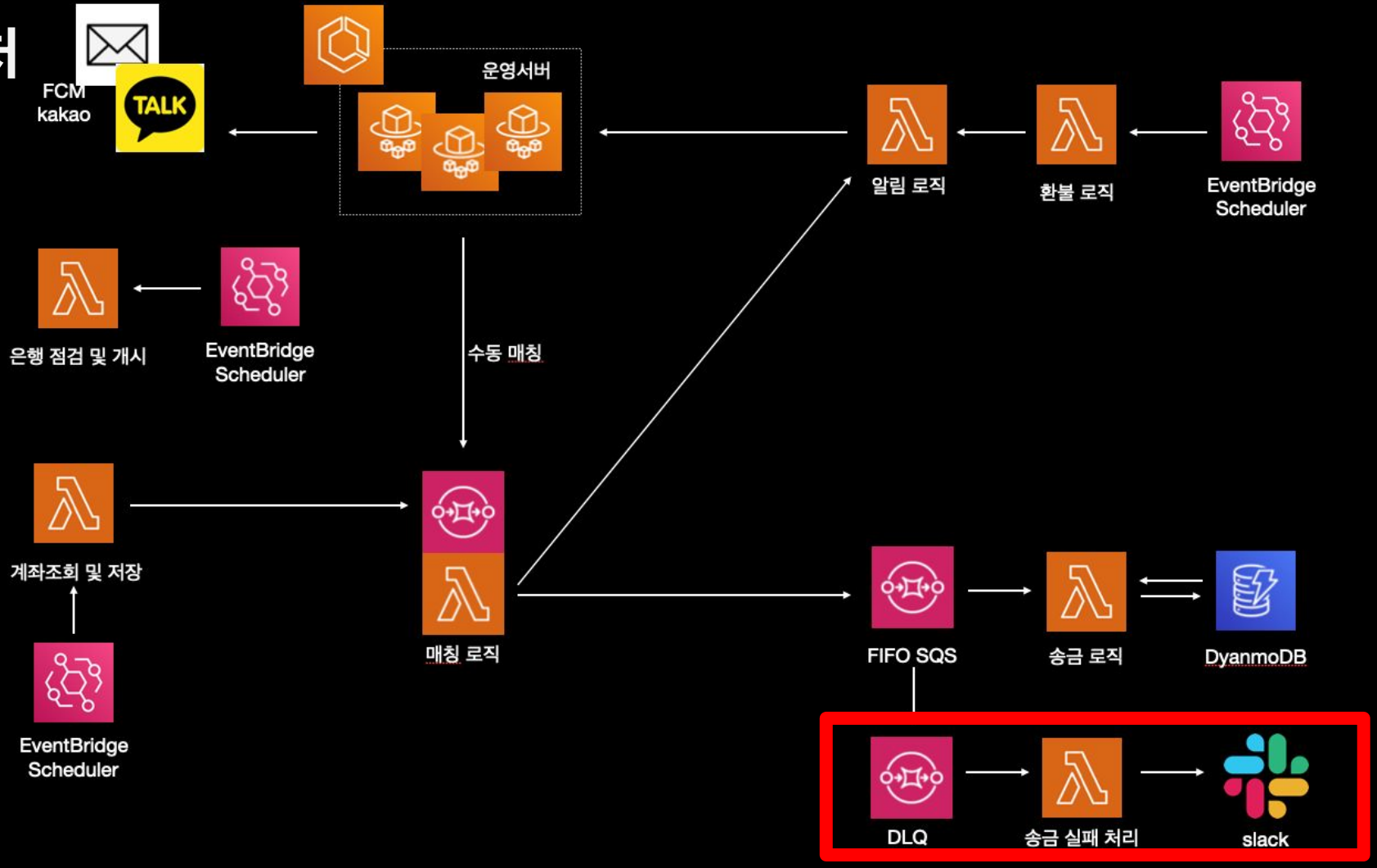
아키텍처


FCM
kakao
TALK
운영서버
알림 로직
환불 로직
EventBridge
Scheduler
은행 점검 및 개시
EventBridge
Scheduler
수동 매칭
계좌조회 및 저장
매칭 로직
EventBridge
Scheduler
FIFO SQS
송금 로직
DyanmoDB
DLQ
송금 실패 처리
slack

아키텍처


FCM
kakao
TALK
운영서버
알림 로직
환불 로직
EventBridge
Scheduler
은행 점검 및 개시
EventBridge
Scheduler
수동 매칭
계좌조회 및 저장
매칭 로직
EventBridge
Scheduler
FIFO SQS
송금 로직
DyanmoDB
DLQ
송금 실패 처리
slack

아키텍처


FCM
kakao
TALK
운영서버
알림 로직
환불 로직
EventBridge
Scheduler
은행 점검 및 개시
EventBridge
Scheduler
수동 매칭
계좌조회 및 저장
매칭 로직
FIFO SQS
송금 로직
DyanmoDB
EventBridge
Scheduler
DLQ
송금 실패 처리
slack

운영 개발 환경 나눠야 하는데..

송금이 안전하게 되어야 할텐데..

나야, SAM

AUSG

AWS SAM(AWS Serverless Application Model)

AUSG

SAM이 뭔가요?

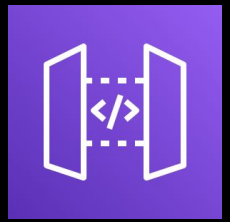
서버리스 애플리케이션을 손쉽게 구축 & 배포할 수 있도록 돕는 프레임워크

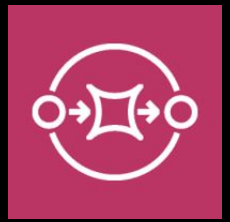
AUSG

SAM이 뭔가요?

서버리스 애플리케이션을 손쉽게 구축 & 배포할 수 있도록 돕는 프레임워크

CloudFormation 기반으로 여러 서버리스 서비스 정의



etc…

SAM은 뭐가 좋아요?

- sam build -> sam deploy 만으로 쉬운 람다 배포
- 🐳 로컬에서 람다 테스트 가능 (sam local invoke)
- Github Action과 쉬운 통합
- IaC 특성을 잘 살려 운영/개발 환경 손쉽게 분리 가능

AUSG

SAM은 이런거에요

sam init

```
sam-app
  events
    event.json
  hello_world
    __init__.py
    app.py
    requirements.txt
  tests
    integration
    unit
    __init__.py
    requirements.txt
  __init__.py
  .gitignore
  README.md
  samconfig.toml
  template.yaml
```

AUSG

SAM은 이런거에요

- sam-app
  - events
    - event.json
  - hello_world
    - __init__.py
    - app.py
    - requirements.txt
  - tests
    - integration
    - unit
    - __init__.py
    - requirements.txt
  - __init__.py
  - .gitignore
  - README.md
  - samconfig.toml
  - template.yaml

AUSG

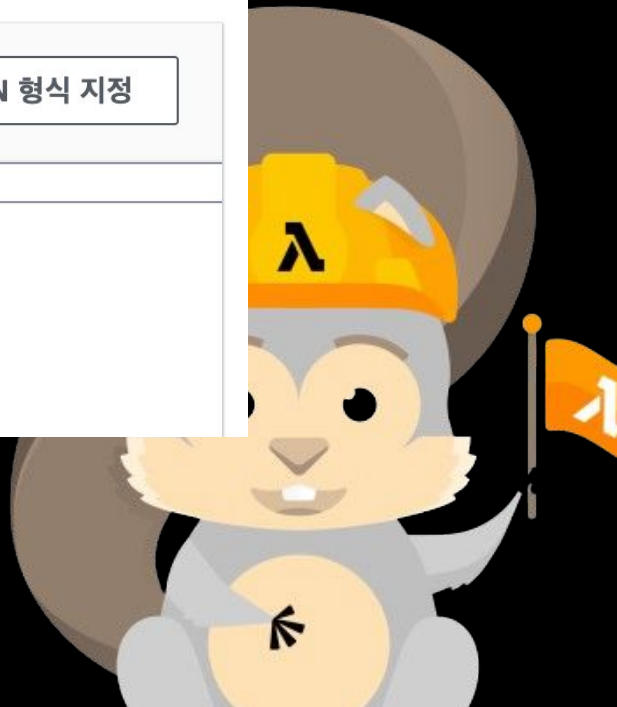
SAM은 이런거에요

- sam-app
  - events
    - {} event.json
  - hello_world
    - __init__.py
  - .gitignore
  - README.md
  - samconfig.toml
  - template.yaml

이벤트 JSON

JSON 형식 지정

```
1 ▾ {
2     "key1": "value1",
3     "key2": "value2",
4     "key3": "value3"
5 }
```



AUSG

SAM은 이런거에요

- sam-app
  - events
    - event.json
  - hello_world
    - __init__.py
    - app.py
    - requirements.txt
  - tests
    - integration
    - unit
    - __init__.py
    - requirements.txt
  - __init__.py
  - .gitignore
  - README.md
  - samconfig.toml
  - template.yaml

AUSG

SAM은 이런거에요

- sam-app
  - events
    - event.json
  - hello_world
    - __init__.py
    - app.py
    - requirements.txt
  - tests
    - integration
    - unit
    - __init__.py
    - requirements.txt
  - __init__.py
  - .gitignore
  - README.md
  - samconfig.toml
  - template.yaml

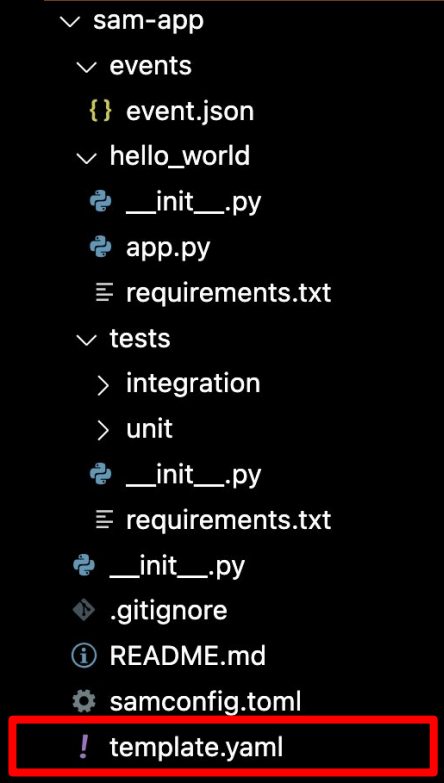
AUSG

SAM은 이런거에요

- sam-app
  - events
    - event.json
  - hello_world
    - __init__.py
    - app.py
    - requirements.txt
  - tests
    - integration
    - unit
    - __init__.py
    - requirements.txt
  - __init__.py
  - .gitignore
  - README.md
  - samconfig.toml
  - template.yaml

AUSG

SAM은 이런거에요

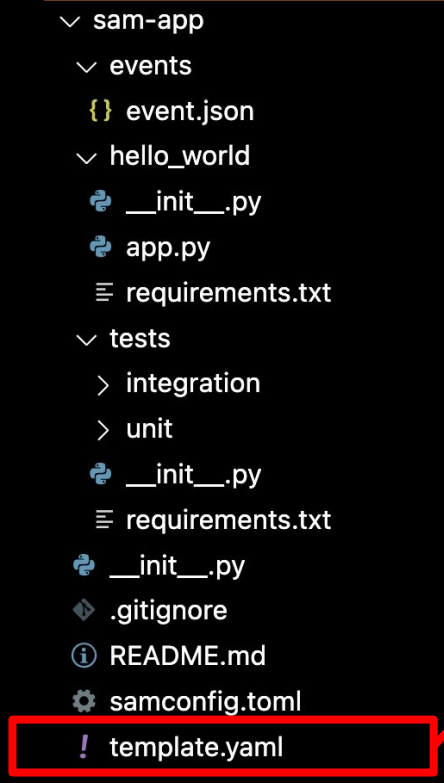


template.yaml

- CloudFormation 템플릿 일종
- 서버리스 애플리케이션 리소스 정의
  - AWS::Severless::Fuction
  - AWS::Severless::Api
  - AWS::Severless::HttpApi
  - AWS::Severless::SimpleTable
- Resoureces에는 기존 CloudFormation 리소스도 가능

AUSG

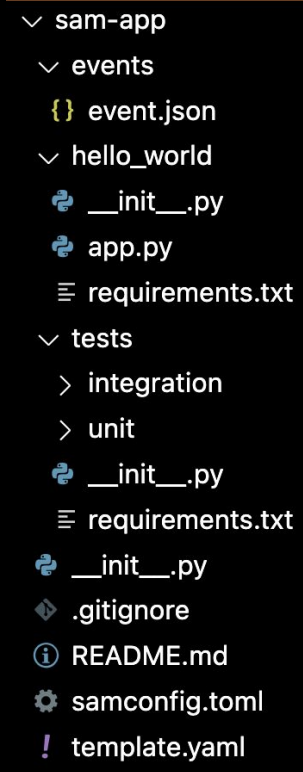
SAM은 이런거에요




```yaml
Globals:
  Function:
    Timeout: 3
    MemorySize: 128

    Tracing: Active
  Api:
    TracingEnabled: true
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: hello_world/
      Handler: app.lambda_handler
      Runtime: python3.9
      Architectures:
      - x86_64
      Events:
        HelloWorld:
          Type: Api
          Properties:
            Path: /hello
            Method: get
```

AUSG

SAM은 이런거에요



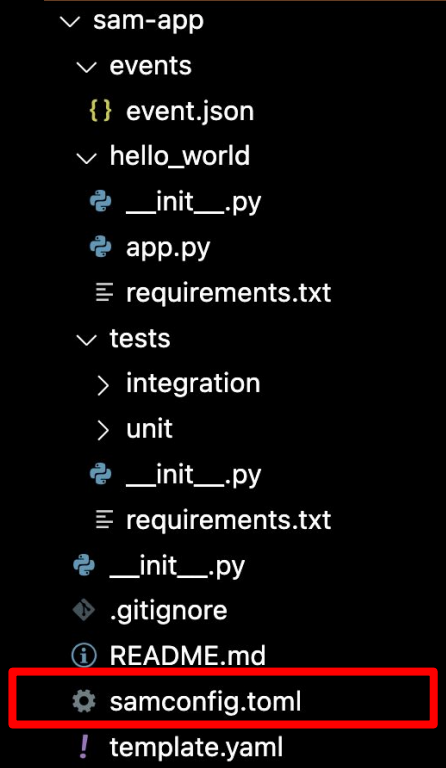

```yaml
Globals:
  Function:
    Timeout: 3
    MemorySize: 128

    Tracing: Active
  Api:
    TracingEnabled: true
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: hello_world/
      Handler: app.lambda_handler
```

```yaml
Events:
  SQSEvent:
    Type: SQS
    Properties:
      Queue: !Sub '{{resolve:ssm:/smart/${NowEnvironment}/transfer-sqs-arn}}'
      BatchSize: 1
      Enabled: true
```

```yaml
                    Method: get
```

AUSG

SAM은 이런거에요

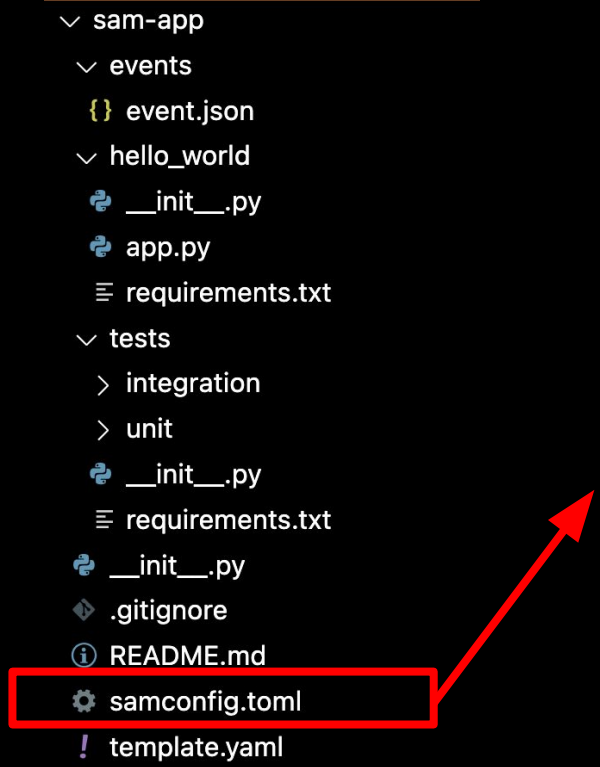


samconfig.toml

- SAM CLI 명령어의 기본 구성을 저장
- sam build, deploy의 설정
- 반복되는 설정 입력을 피할 수 있다.

AUSG

SAM은 이런거에요




```
[default]
[default.global.parameters]
stack_name = "sam-app"

[default.build.parameters]
cached = true
parallel = true

[default.validate.parameters]
lint = true

[default.deploy.parameters]
capabilities = "CAPABILITY_IAM"
confirm_changeset = true
resolve_s3 = true

[default.package.parameters]
resolve_s3 = true

[default.sync.parameters]
watch = true

[default.local_start_api.parameters]
warm_containers = "EAGER"
```

AUSG

SAM으로 개발 운영 환경 분리하기

samconfig.toml

```toml
[dev]
[dev.global]
[dev.global.parameters]
stack_name = "dev-test"

[dev.deploy]
[dev.deploy.parameters]
s3_prefix = "dev-test"
region = "ap-northeast-2"
resolve_s3 = true
disable_rollback = true
image_repositories = []
capabilities = "CAPABILITY_IAM"
confirm_changeset = true
parameter_overrides = [
    "Environment=dev"
]
```

```toml
[prod]
[prod.global]
[prod.global.parameters]
stack_name = "prod-test"

[prod.deploy]
[prod.deploy.parameters]
s3_prefix = "prod-test"
region = "ap-northeast-2"
resolve_s3 = true
disable_rollback = true
image_repositories = []
capabilities = "CAPABILITY_IAM"
confirm_changeset = true
parameter_overrides = [
    "Environment=prod"
]
```

AUSG

SAM으로 개발 운영 환경 분리하기

samconfig.toml

[dev]와 [prod] 분리

```toml
[dev]
[dev.global]
[dev.global.parameters]
stack_name = "dev-test"

[dev.deploy]
[dev.deploy.parameters]
s3_prefix = "dev-test"
region = "ap-northeast-2"
resolve_s3 = true
disable_rollback = true
image_repositories = []
capabilities = "CAPABILITY_IAM"
confirm_changeset = true
parameter_overrides = [
    "Environment=dev"
]
```

```toml
[prod]
[prod.global]
[prod.global.parameters]
stack_name = "prod-test"

[prod.deploy]
[prod.deploy.parameters]
s3_prefix = "prod-test"
region = "ap-northeast-2"
resolve_s3 = true
disable_rollback = true
image_repositories = []
capabilities = "CAPABILITY_IAM"
confirm_changeset = true
parameter_overrides = [
    "Environment=prod"
]
```

AUSG

SAM으로 개발 운영 환경 분리하기

samconfig.toml

[dev]와 [prod] 분리

```
[dev]
[dev.global]
[dev.global.parameters]
stack_name = "dev-test"

[dev.deploy]
[dev.deploy.parameters]
s3_prefix = "dev-test"
region = "ap-northeast-2"
resolve_s3 = true
disable_rollback = true
image_repositories = []
capabilities = "CAPABILITY_IAM"
confirm_changeset = true
parameter_overrides = [
    "Environment=dev"
]
```

```
[prod]
[prod.global]
[prod.global.parameters]
stack_name = "prod-test"

[prod.deploy]
[prod.deploy.parameters]
s3_prefix = "prod-test"
region = "ap-northeast-2"
resolve_s3 = true
disable_rollback = true
image_repositories = []
capabilities = "CAPABILITY_IAM"
confirm_changeset = true
parameter_overrides = [
    "Environment=prod"
]
```

sam deploy –config-env

AUSG

SAM으로 개발 운영 환경 분리하기

samconfig.toml

```
[dev]
[dev.global]
[dev.global.parameters]
stack_name = "dev-test"

[dev.deploy]
[dev.deploy.parameters]
s3_prefix = "dev-test"
region = "ap-northeast-2"
resolve_s3 = true
disable_rollback = true
image_repositories = []
                                    IAM"
confirm_changeset = true
parameter_overrides = [
    "Environment=dev"
]
```

```
[prod]
[prod.global]
[prod.global.parameters]
stack_name = "prod-test"

[prod.deploy]
[prod.deploy.parameters]
s3_prefix = "prod-test"
region = "ap-northeast-2"
resolve_s3 = true
disable_rollback = true
image_repositories = []
capabilities = "CAPABILITY_IAM"
confirm_changeset = true
parameter_overrides = [
    "Environment=prod"
]
```

template.yaml의 Parameter를 오버라이딩

잠깐! Parameters가 뭐에요?

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31

Parameters:
  Environment:
    Type: String
```

- CloudFormation 변수 선언 유사
- !Ref나 !Sub로 탬플릿 내에서 파라미터 값 참조 가능

AUSG

SAM으로 람다 잘 쓰기

samconfig.toml

```
[dev]
[dev.global]
[dev.global.parameters]
stack_name = "dev-test"

[dev.deploy]
[dev.deploy.parameters]
s3_prefix = "dev-test"
region = "ap-northeast-2"
resolve_s3 = true
disable_rollback = true
image_repositories = []
                              IAM"
parameter_overrides = [
    "Environment=dev"
]
```

```
[prod]
[prod.global]
[prod.global.parameters]
stack_name = "prod-test"

[prod.deploy]
[prod.deploy.parameters]
s3_prefix = "prod-test"
region = "ap-northeast-2"
resolve_s3 = true
disable_rollback = true
image_repositories = []
capabilities = "CAPABILITY_IAM"
confirm_changeset = true
parameter_overrides = [
    "Environment=prod"
]
```

template.yaml의 Parameter를 오버라이딩

AUSG

template.yaml

```yaml
Resources:
  TransferFunction:
    Type: AWS::Serverless::Function
    Properties:
      FunctionName: !Sub "${Environment}-test"
      CodeUri: test/
      Handler: app.lambda_handler
      Runtime: python3.9
      Architectures:
      - x86_64
      Policies:
      - AmazonSQSFullAccess
      - SSMParameterReadPolicy:
          ParameterName: '/example/*'
      Events:
        SQSEvent:
          Type: SQS
          Properties:
            Queue: !Sub '{{resolve:ssm:/example/${Environment}/sqs-url}}'
            BatchSize: 5
            Enabled: true
      Environment:
        Variables:
          DB_URL: !Sub '{{resolve:ssm:/example/${Environment}/db-url}}'
          ENV: !Ref Environment
```

!Sub로 문자열에 Environment 값 삽입 가능

AUSG

template.yaml

```yaml
Resources:
  TransferFunction:
    Type: AWS::Serverless::Function
    Properties:
      FunctionName: !Sub "${Environment}-test"
      CodeUri: test/
      Handler: app.lambda_handler
      Runtime: python3.9
      Architectures:
      - x86_64
      Policies:
      - AmazonSQSFullAccess
      - SSMParameterReadPolicy:
          ParameterName: '/example/*'
      Events:
        SQSEvent:
          Type: SQS
          Properties:
            Queue: !Sub '{{resolve:ssm:/example/${Environment}/sqs-url}}'
            BatchSize: 5
            Enabled: true
      Environment:
        Variables:
          DB_URL: !Sub '{{resolve:ssm:/example/${Environment}/db-url}}'
          ENV: !Ref Environment
```

!Sub로 문자열에 Environment 값 삽입 가능

AUSG

SAM으로 개발 운영 환경 분리하기

template.yaml

```yaml
Environment:
  Variables:
    DB_URL: !Sub '{{resolve:ssm:/example/${Environment}/db-url}}'
    ENV: !Ref Environment
```

!Ref로 Parameter 값을 그대로 사용 가능

AUSG

SSM Parameter Store로 공통 변수 관리

와! 드디어 환경 분리 완료!

AUSG

SSM Parameter Store로 공통 변수 관리

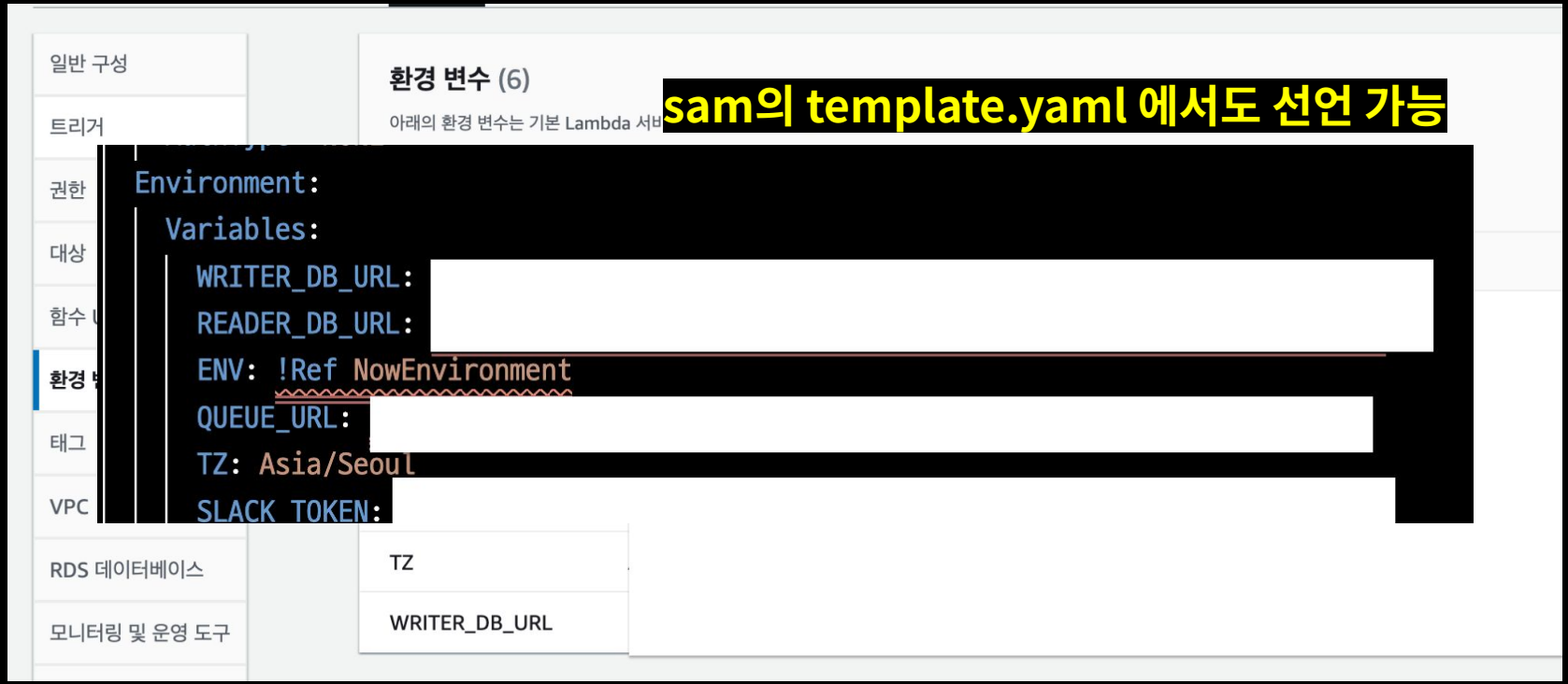
근데 환경 변수.. 어떻게 관리하지?

AUSG

SSM Parameter Store로 공통 변수 관리

AUSG

SSM Parameter Store로 공통 변수 관리



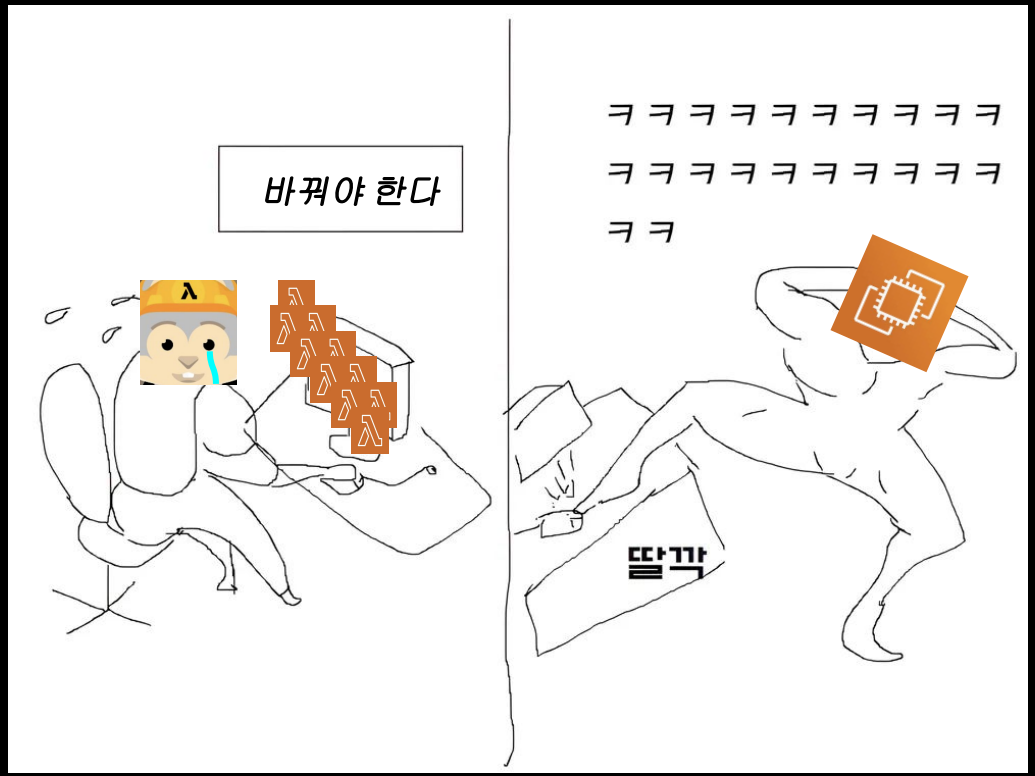
AUSG

SSM Parameter Store로 공통 변수 관리

람다도 많고.. 공통 변수도 많은데..
만약 변수 하나가 변경되면 …?

AUSG

SSM Parameter Store로 공통 변수 관리



AUSG

SSM Parameter Store로 공통 변수 관리

AWS Systems Manager Parameter Store

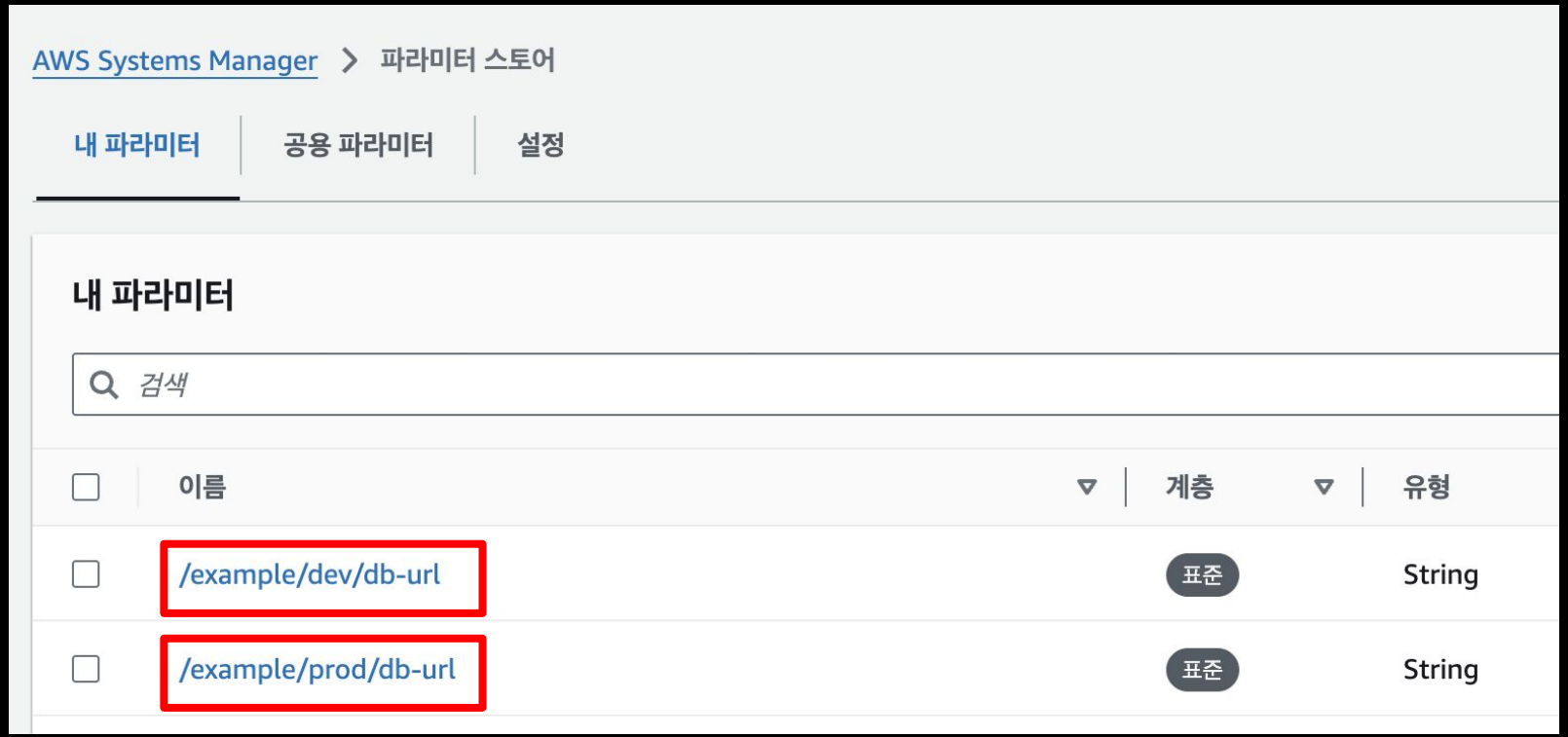
SSM Parameter Store로 공통 변수 관리

AWS Systems Manager Parameter Store

- 키 값 파라미터 저장소
- CloudFormation, Lambda, EC2 등 다른 AWS 서비스와 쉽게 통합
- KMS와도 통합하여 쉽게 암호화 가능!

AUSG

SSM Parameter Store로 공통 변수 관리



AUSG

SSM Parameter Store로 공통 변수 관리

```yaml
Resources:
  TestFunction:
    Type: AWS::Serverless::Function
    Properties:
      FunctionName: !Sub "${Environment}-test"
      CodeUri: test/
      Handler: app.lambda_handler
      Runtime: python3.9
      Architectures:
      - x86_64
      Policies:
      - SSMParameterReadPolicy:
          ParameterName: '/example/*'
```

AUSG

SSM Parameter Store로 공통 변수 관리

```
Environment:
  Variables:
    DB_URL: !Sub '{{resolve:ssm:/example/${Environment}/db-url}}'
    ENV: !Ref Environment
```

{{resolve:ssm:경로}} 를 통해서 sam deploy시 CloudFormation가 값을 주입

SSM Parameter Store로 공통 변수 관리

쓰면 좋은 점

- 모든 람다의 환경변수를 각각 변경하지 않아도 됨
- 중앙에서 관리하기 때문에 편함

한계점

- SAM에서 아직까지도 보안 문자열 파라미터는 지원하지 않음
- 파라미터를 바꾸더라도 함수를 재배포 해야함
  - 스크립트 제작
  - boto3의 ssm.get_parameter를 통해서 동적으로 주입 검토

AUSG

Github Action과 통합

와! 환경변수도 일단락!

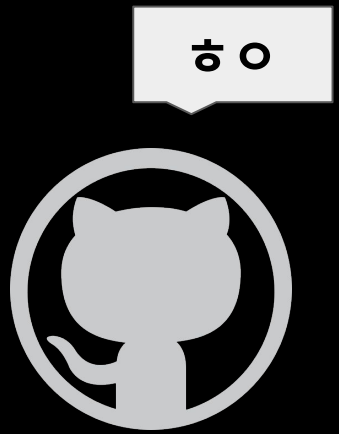
근데 변경사항이 있을 때 마다

sam build -> sam deploy --config-env dev -> sam deploy --config-env prod ..

뭔가 불편한데?

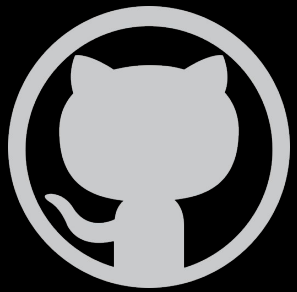
AUSG

Github Action과 통합




```yaml
name: Deploy SAM Application

on:
  push:
    branches:
      - develop
      - master

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Check out code
        uses: actions/checkout@v2

      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: '3.9'

      - name: Install AWS SAM CLI
        run: |
          pip install aws-sam-cli

      - name: Set environment
        run: |
          if [ $ == 'refs/heads/master' ]; then
            echo "SAM_ENV=prod" >> $GITHUB_ENV
          else
            echo "SAM_ENV=dev" >> $GITHUB_ENV
          fi
```

Github Action과 통합



```yaml
name: Deploy SAM Application

on:
  push:
    branches:
      - develop
      - master
```

AUSG

Github Action과 통합

브랜치에 따라서 $GITHUB_ENV를 다르게

```yaml
jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Check out code
        uses: actions/checkout@v2

      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: '3.9'

      - name: Install AWS SAM CLI
        run: |
          pip install aws-sam-cli

      - name: Set environment
        run: |
          if [ $ == 'refs/heads/master' ]; then
            echo "SAM_ENV=prod" >> $GITHUB_ENV
          else
            echo "SAM_ENV=dev" >> $GITHUB_ENV
          fi
```

AUSG

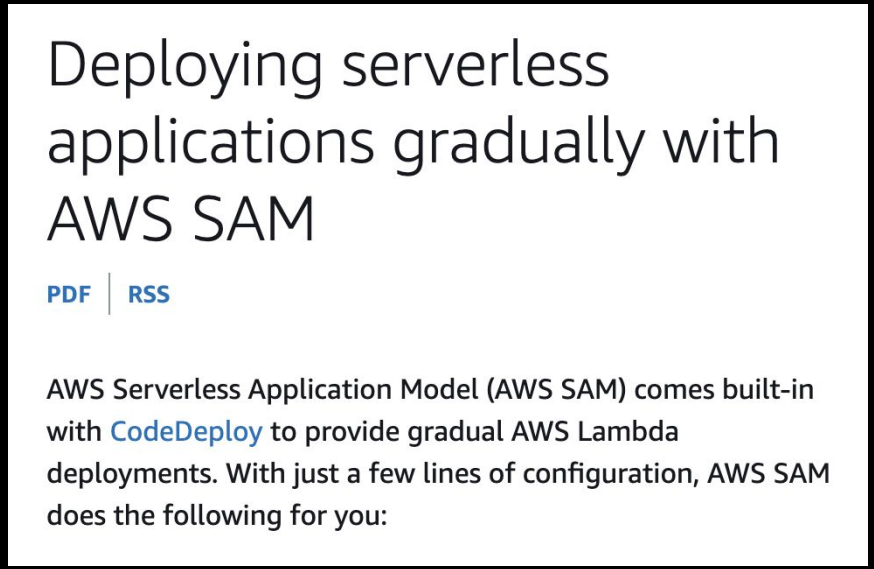
Github Action과 통합

```
- name: Build SAM application
  run: |
    sam build --config-env ${{ env.SAM_ENV }}

- name: Deploy SAM application
  env:
    AWS_ACCESS_KEY_ID:
    AWS_SECRET_ACCESS_KEY:
    AWS_REGION: ap-northeast-2
  run: |
    sam deploy --config-env ${{ env.SAM_ENV }} --no-confirm-changeset --capabilities CAPABILITY_IAM
```

${{env.}} 을 사용해서 prod면 prod, dev면 dev를 삽입

AUSG

카나리아/리니어 배포



배포시 다운타임 없이 안전하게 배포해보자!

AUSG

카나리아/리니어 배포

Deployment Preference Type
Canary10Percent30Minutes
Canary10Percent5Minutes
Canary10Percent10Minutes
Canary10Percent15Minutes
Linear10PercentEvery10Minutes
Linear10PercentEvery1Minute
Linear10PercentEvery2Minutes
Linear10PercentEvery3Minutes
AllAtOnce

AUSG

카나리아/리니어 배포

Deployment Preference Type
Canary10Percent30Minutes
Canary10Percent5Minutes
Canary10Percent10Minutes
Canary10Percent15Minutes
Linear10PercentEvery10Minutes
Linear10PercentEvery1Minute
Linear10PercentEvery2Minutes
Linear10PercentEvery3Minutes
AllAtOnce

```yaml
Parameters:
  Environment:
    Type: String

Conditions:
  IsProd: !Equals [!Ref Environment, prod]
```

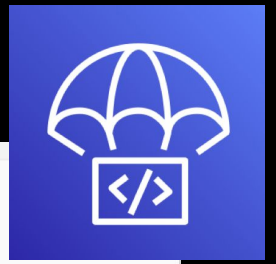
AUSG

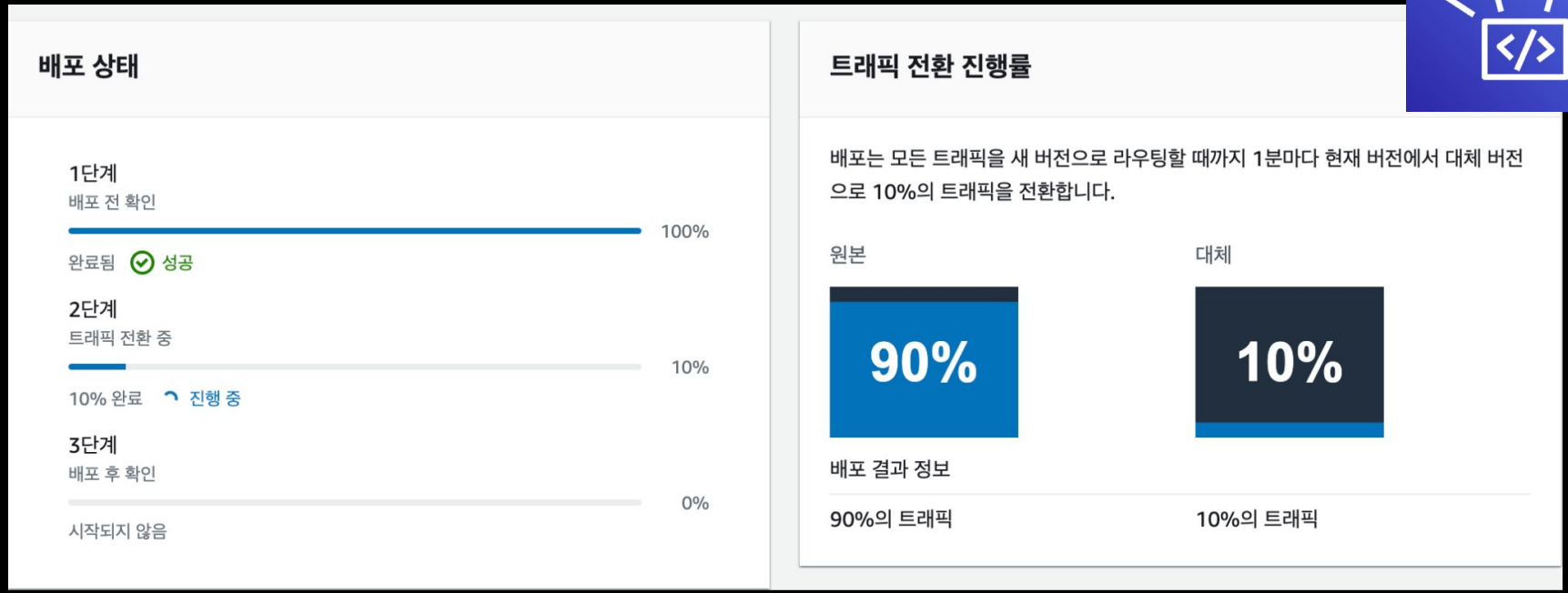
카나리아/리니어 배포

```yaml
Parameters:
  Environment:
    Type: String

Conditions:
  IsProd: !Equals [!Ref Environment, prod]

Properties:
  FunctionName: !Sub "${NowEnvironment}-test"
  CodeUri: test/
  Handler: app.lambda_handler
  Runtime: python3.9
  Architectures:
  - x86_64
  AutoPublishAlias: live
  DeploymentPreference:
    Type:
      - !If [IsProd, Linear10PercentEvery1Minute, AllAtOnce]
```

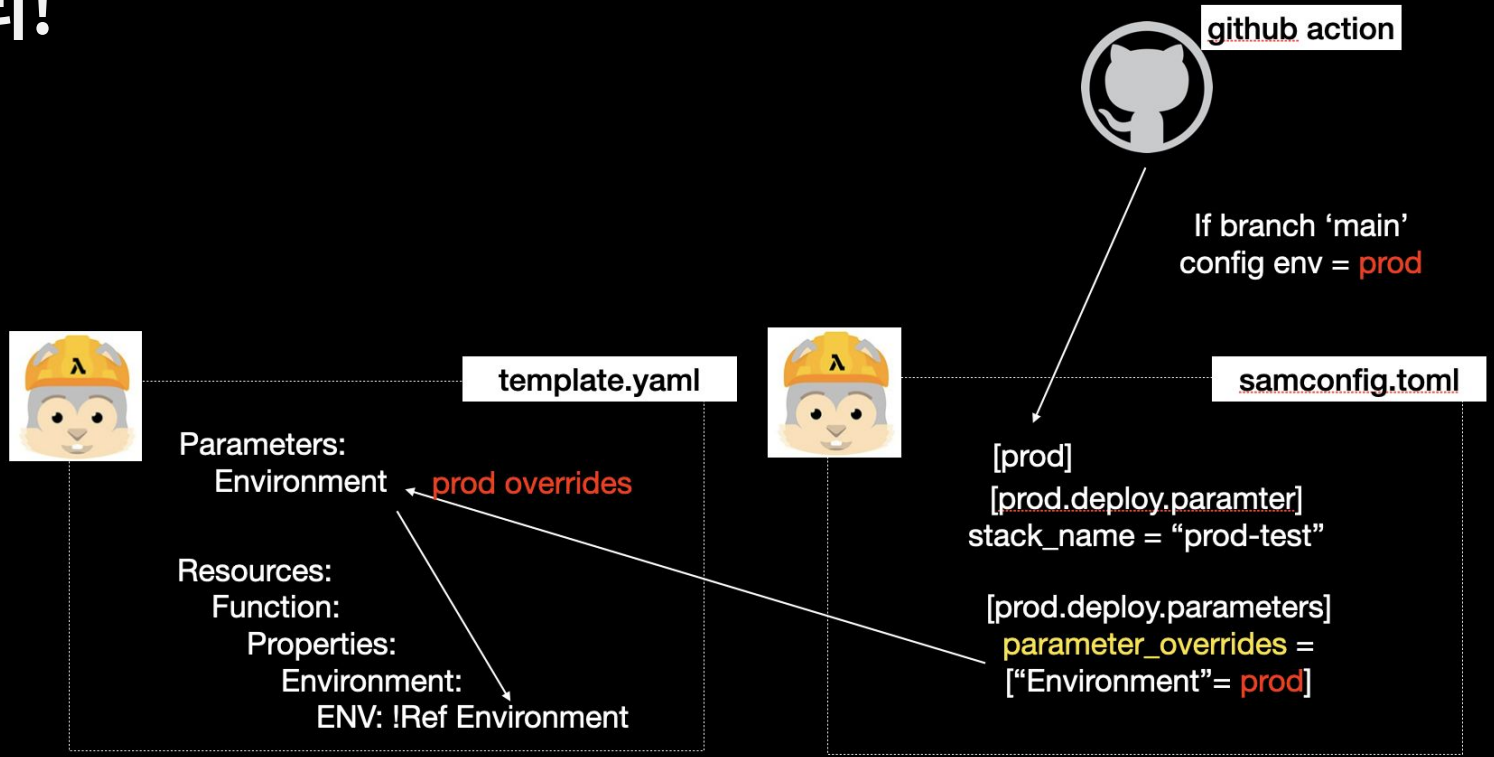
AUSG

카나리아/리니어 배포





정리!


github action
If branch 'main'
config env = prod
template.yaml
samconfig.toml
Parameters:
Environment
prod overrides
Resources:
Function:
Properties:
Environment:
ENV: !Ref Environment
[prod]
[prod.deploy.paramter]
stack_name = "prod-test"
[prod.deploy.parameters]
parameter_overrides =
["Environment"= prod]

정리!

SSM Parameter Store

template.yaml

파라미터 주입

```
Environment:
    ENV: !Ref Environment
    DB_URL: {{resolve:ssm:경로}}
```

```
Conditions:
    IsProd: !Equals [!Ref Environment, prod]
```

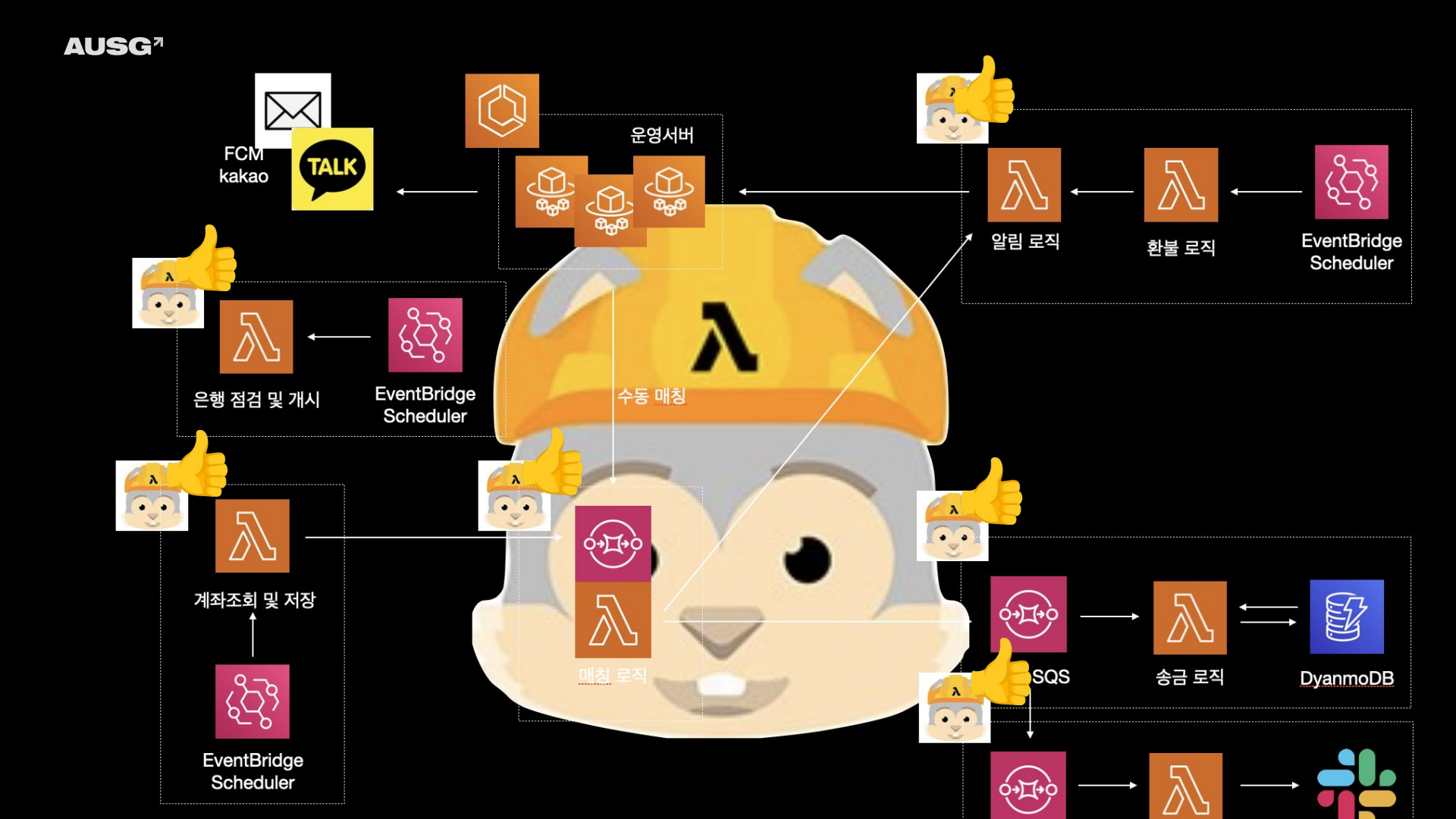
IsProd = True

```
DeploymentPreference
    Type:
        - !If [IsProd, Linear10Percent ..., AllAtOnce]
```

CodeDeploy

배포 완료

AUSG


FCM
kakao
TALK
운영서버
알림 로직
환불 로직
EventBridge
Scheduler
은행 점검 및 개시
EventBridge
Scheduler
수동 매칭
계좌조회 및 저장
EventBridge
Scheduler
매칭 로직
SQS
송금 로직
DyanmoDB

AUSG↗

<세 줄 요약>

- Lambda로 송금 시스템을 올렸다.
- SAM과 SSM Parameter Store와 Github Action으로 광명을 찾았다.
- 점진적 배포를 통해서 안정적인 서비스 운영이 가능했다.

SQS도 핵심이었는데 20분은 너무 짧다.

AUSG

감사합니다.